Mathematical formulations for scheduling jobs on identical parallel machines with family setup times and total weighted completion time minimization

Arthur Kramer[a,], Manuel Iori[a], Philippe Lacomme[b]

[a]*DISMI, Università degli Studi di Modena e Reggio Emilia*
Via Amendola 2, Pad. Morselli, 42122 Reggio Emilia, Italia
[b]*ISIMA, Université Clermont Auvergne, 1 rue de la Chebarde, 63178 Aubière, France*

Abstract

This paper addresses the parallel machine scheduling problem with family dependent setup times and total weighted completion time minimization. In this problem, when two jobs j and k are scheduled consecutively on the same machine, a setup time is performed between the finishing time of j and the starting time of k if and only if j and k belong to different families. The problem is strongly $\mathcal{NP}$-hard and is commonly addressed in the literature by heuristic approaches and by branch-and-bound algorithms. Achieving proven optimal solution is a challenging task even for small size instances. Our contribution is to introduce five novel mixed integer linear programs based on concepts derived from one-commodity, arc-flow and set covering formulations. Numerical experiments on more than 10000 benchmark instances show that one of the arc-flow models and the set covering model are quite efficient, as they provide on average better solutions than state-of-the-art approaches with shorter computation times, and solve to proven optimality a large number of open instances from the literature.

Keywords: Scheduling, mathematical formulations, parallel machines, family setup times, weighted completion time

1. Introduction

Let $J = \{1, \ldots, n\}$ be a set of jobs to be scheduled on a set $M = \{1, \ldots, m\}$ of identical parallel machines, with $m \leq n$. Each job $j \in J$ has an integer processing time p_j and an integer penalty weight w_j. In addition, each job $j \in J$ belongs to a family

Email addresses: arthur.kramer@unimore.it (Arthur Kramer), manuel.iori@unimore.it (Manuel Iori), placomme@isima.fr (Philippe Lacomme)

$i \in F = \{1, \ldots, f\}$, with $f \leq n$, so the set of jobs can be partitioned as $J = \cup_{i \in F} J_i$, in such a way that each set J_i contains the jobs of family $i \in F$. Let $i(j)$ denote the index of the family of job j. An integer setup time s_i is associated with each family $i \in F$, meaning that if a job j is processed on a machine immediately after a job k and $i(j) \neq i(k)$, then a setup of s_i units of time must be performed before starting processing j. Preemption is not allowed, so once the processing of a job is started it cannot be interrupted. The objective is to schedule all jobs on the machines by minimizing the sum of their weighted completion times. In the three field classification of Graham et al. (1979), this problem is referred to as $P|s_i|\sum w_j C_j$, where C_j stands for the completion time of job j. The $P|s_i|\sum w_j C_j$ is a strongly $\mathcal{NP}$-hard problem, as the simpler problem with unitary weights is known to be $\mathcal{NP}$-hard (see Webster 1997).

The problem is interesting because models many real-world situations, where some activities must be performed by a set of agents in parallel, and each activity has a certain level of importance expressed by the weight. Many production problems can be indeed modeled as a $P|s_i|\sum w_j C_j$, and in such cases the weight usually defines a monetary importance of a job. Other relevant applications arise in the context of health-care, where, for example, patients have to be assigned to surgery rooms that must be equipped by considering the type (i.e., the family) of surgery to be performed. In such cases, the weight usually models a level of urgency for the patient. As stressed in the surveys of Potts and Kovalyov (2000), Allahverdi et al. (2008) and Allahverdi (2015), the $P|s_i|\sum w_j C_j$ has not received much attention in the recent literature. To our knowledge, the most recent works on parallel machine scheduling problem with family setup and total weighted completion time minimization are the ones of Liao et al. (2012) and Tseng and Lee (2017). The former article proposes an heuristic that combines elements from tabu search, least loaded processor rule and the comparison with schedules without setups, whereas the latter proposes an electromagnetism-like based metaheuristic. Regarding exact methods, mainly *branch-and-bound* (B&B) algorithms have been developed, and the last relevant papers date back to the early 2000s (Webster and Azizoglu, 2001; Chen and Powell, 2003; Dunstall and Wirth, 2005b; Bettayeb et al., 2008). According to Allahverdi et al. (2008) and Allahverdi (2015), the papers of Chen and Powell (2003) and Dunstall and Wirth (2005b) still represent the state-of-the-art exact approaches to solve the $P|s_i|\sum w_j C_j$.

Some closely related problems, such as the $1||\sum C_j$, the $1||\sum w_j C_j$, the $P||\sum C_j$ and the $P||\sum w_j C_j$ are well studied in the literature. The first three problems are polynomially solvable by the (weighted) Smith's ratio rule.The $P||\sum w_j C_j$, instead, was proven to be $\mathcal{NP}$-hard by Bruno et al. (1974). Kramer et al. (2018) recently addressed this problem by

means of an *arc-flow* (AF) formulation that models the sequences of jobs on the machines as paths in a network of pseudo-polynomial size. The AF formulation solved instances with up to 1000 jobs in a few minutes of computing effort, consistently improving previous results in the literature. These considerations motivated us to propose new mathematical formulations and exact methods, as a tentative to fill the lack in the literature for exact methods for the $P|s_i|\sum w_jC_j$. In particular, we propose and investigate five *mixed integer linear programs* (MILPs). All proposed formulations are novel and include different ways to tackle the difficulties of the problem at hand. The first one is a *one-commodity* (OC) formulation that has polynomial size in the number of jobs. The next three MILPs are AF models that formulate the problem by making use of pseudo-polynomial numbers of variables and constraints. The last one is a *set covering* (SC) formulation that uses an exponential number of variables. Enumerating all variables in the SC model is unpractical, so we solve it by a tailored *branch-and-price* (B&P) approach. In addition to that, we also investigate simple but effective valid inequalities.

We performed extensive computational experiments on sets of benchmark instances from the literature. As shown below, our results prove that one of the AF formulations and the SC one are able to solve to proven optimality several open instances from the literature. Despite these good results, the problem remains very challenging, especially because of the family setup times. All benchmark instances of the $P||\sum w_jC_j$ with up to 400 jobs were solved to proven optimality by Kramer et al. (2018) with an AF. Despite our attempts, instances of the $P|s_i|\sum w_jC_j$ with just 40 jobs remain unsolved.

The remainder of this work is organized as follows. A concise review of the literature on the $P|s_i|\sum w_jC_j$ and closely related problems is provided in Section 2. The newly proposed mathematical formulations are introduced and described in Section 3. The extensive computational experiments that we performed are presented and discussed in detail in Section 4, then Section 5 reports the concluding remarks and future research directions.

2. Brief literature review

Despite its importance in practice, the $P|s_i|\sum w_jC_j$ has not received much attention in recent years. Nevertheless, interesting theoretical results have been achieved in the 90s and early 2000s. Many of these results were attained by extending known properties of closely related problems, such as the $1||\sum C_j$, the $P||\sum C_j$, the $1||\sum w_jC_j$, the $P||\sum w_jC_j$ and the $1|s_i|\sum w_jC_j$. All these problems have in common the fact that they can benefit from the well known Smith's rule, also known as *shortest processing time* (SPT) rule. The first and the second problems are polynomially solvable by the direct application of the SPT. The

$1||\sum w_jC_j$ in turn, is efficiently solved by a modified version of the SPT, denoted *weighted shortest processing time* (WSPT), in which jobs are sorted by non-decreasing order of their p_j/w_j ratio. For what concerns the $P||\sum w_jC_j$, the $1|s_i|\sum w_jC_j$ and the $P|s_i|\sum w_jC_j$ itself, we mention that the first and the last problem were proven $\mathcal{NP}$-hard by Bruno et al. (1974) and Webster (1997), respectively, while the second can be solved in $O(f^2n^f)$ by a *dynamic programming* (DP) algorithm proposed by Ghosh (1994). It is well-known (see, e.g, Elmaghraby and Park 1974) that there exists an optimal solution of the $P||\sum w_jC_j$ where the sequences of jobs on each machine follow the WSPT order. Numerous algorithms take advantage of this property. Among these, we highlight the B&B methods of Azizoglu and Kirca (1999), Chen and Powell (1999) and van den Akker et al. (1999) (improved very recently by Kowalczyk and Leus 2018), as well as the enhanced AF formulations by Kramer et al. (2018).

The WSPT rule is also useful for the $1|s_i|\sum w_jC_j$. Monma and Potts (1989) showed that in an optimal solution jobs of the same family must be scheduled according the WSPT order. Mason and Anderson (1991) proved that there exists an optimal solution where the batches appear in the *shortest weighted mean processing time* (SWMPT) order. Some B&B and heuristic algorithms based on these properties were proposed by Crauwels et al. (1997, 1998) and Dunstall et al. (2000).

The above results on the $1|s_i|\sum w_jC_j$ were also extended to the parallel machine case, where for an optimal solution the sequences on any machine must fulfill the two mentioned conditions. Webster and Azizoglu (2001) proposed two DP algorithms to solve the $P|s_i|\sum w_jC_j$. When the number of families and machines are fixed, their methods are polynomial in the sum of the job weights and in the sum of the family setup and job processing times, respectively. Azizoglu and Webster (2003) and Dunstall and Wirth (2005a) developed B&B algorithms. These B&Bs are based on list-scheduling techniques, where partial schedules are constructed at each node of the B&B tree, and on lower bounding techniques to solve $1|s_i|\sum w_jC_j$ and $P||\sum w_jC_j$ subproblems. These approaches have been able to solve instances with up to 25 jobs. Chen and Powell (2003) also tackled the problem, but by means of a B&P algorithm in which each node is solved by invoking a *column generation* (CG) algorithm. By using this approach, the authors were able to solve some instances with up to 40 jobs and 6 machines. More recently, Bettayeb et al. (2008) developed a B&B method that relies on tighter lower bounds than those adopted by Dunstall and Wirth (2005a) and is able to reduce the number of explored nodes, although at the expense of higher computational times. This method could solve some instances with 45 jobs and limited number of families and machines.

Heuristic and metaheuristic methods were also proposed in the literature. In the work of Dunstall and Wirth (2005b) multiple heuristic methods based on list-scheduling, single machine subproblems and improvement phases were designed. The authors evaluated their methods by comparison with exact methods and lower bounds from the literature on instances containing up to 80 jobs, 5 machines and 16 families. They showed that their methods were able to provide small optimality gaps for all such instances. More recently, Liao et al. (2012) and Tseng and Lee (2017) developed new heuristic and metaheuristic approaches, respectively. The approach by Liao et al. (2012) is based on a tabu search that solves parallel machine subproblems without setups with the SWMPT rule, while Tseng and Lee (2017) proposed an electromagnetism-like based metaheuristic that relies on the attraction-repulsion mechanism from the electromagnetic theory. In both works, the performances of the methods were evaluated on benchmark instances with up to 80 jobs.

3. Mathematical formulations

This section presents the novel mathematical models that we developed for solving the $P|s_i|\sum w_jC_j$. It also includes some simple but effective strengthening constraints for one of the AF models.

Before presenting our formulations, let us introduce a simple example, called *Example* 1 in the following, with 6 jobs divided into 3 families to be scheduled on 2 machines. The input data of Example 1 are given in Table 1, where $T = 26$ represents a valid upper bound on the completion time of the activities. An optimal solution of value $z^* = 2 \times 11 + 1 \times 7 + 3 \times 5 + 3 \times 12 + 4 \times 7 + 2 \times 20 = 148$ is provided in Figure 1.

Families		Jobs			
i	s_i	j	p_j	w_j	w_j/p_j
1	2	1	4	2	2.00
		2	2	1	2.00
		3	3	3	1.00
2	4	4	5	3	1.67
		5	3	4	0.75
3	3	6	6	2	3.00

Table 1: Example 1 - input data ($n = 6$, $f = 3$, $m = 2$, $T = 26$)

Note that in the depicted optimal solution jobs belonging to the same family are scheduled consecutively on the same machine, but this is not mandatory. This situation is likely

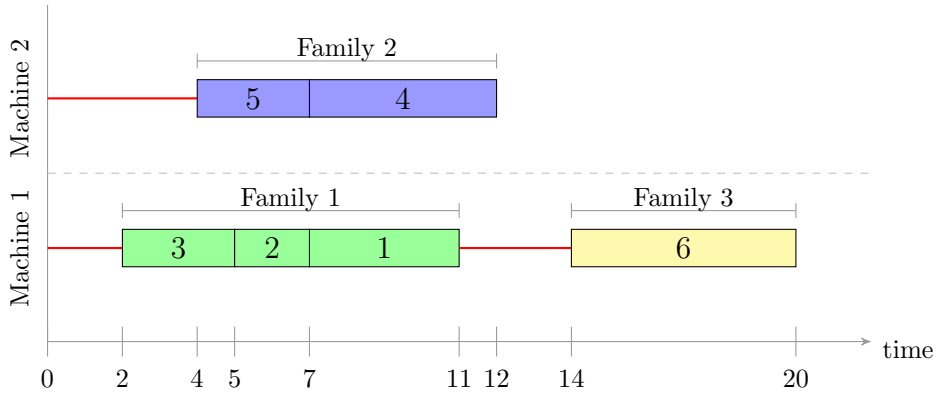

Figure 1: Example 1 - an optimal solution ($z^* = 148$)

to occur often once it allows avoiding the use of setup times. A family setup is only performed between successive jobs on the same machine if the jobs belong to different families. In the solution, a family setup of 3 units of time is added between job 1 and job 6, both processed on machine 1. We notice that before starting processing the first job on each machine, its family setup must be performed. Let us also notice that the batches of consecutive jobs of the same family are scheduled considering the WSPT rule, which is compliant with the propositions of Monma and Potts (1989) and Dunstall and Wirth (2005a).

The value $T = 26$ in Example 1 represents a time horizon estimation, i.e., an upper bound on the completion time of all activities (makespan). The value of T can be estimated by using Property 2 of Chen and Powell (2003) as follows. For each job j, we compute an upper bound T_j on its completion time by first determining

$$\alpha_j = \left\lfloor \frac{1}{m} \left(\sum_{k \in J} (p_k + s_{i(k)}) + (m-1)(p_j + s_{i(j)}) \right) \right\rfloor,$$
$$\beta_j = \sum_{k \in J} (p_k + s_{i(k)}) - \sum_{k \in J_{i(j)} \setminus K_{i,j}} (p_k + s_{i(j)})$$

with $K_{i,j}$ representing the set of jobs of family $i(j)$ that appear before job j in the WSPT order, where $i(j)$ represents the family of job j. Then T_j is simply obtained by setting $T_j = \min(\alpha_j, \beta_j)$, for $j \in J$, and the time horizon value T is calculated as $T = \max_{j \in J}\{T_j\}$. In the remainder of this work, the value of T is computed in this way for all the developed formulations.

3.1. One commodity formulation

Our OC formulation relies on the idea first introduced by Gavish and Graves (1978) for the traveling salesman problem, and widely used since then to formulate many variants of scheduling (e.g., Unlu and Mason 2010), traveling salesman (e.g., Salazar-González and Santos-Hernández 2015) and vehicle routing problems (e.g.,Bruck and Iori 2017), among many. Essentially, the idea is to use a flow variable that models the time when ending the process of a job and starting a new one.

Let us build a graph $G = (V, A)$, with $V = J \cup \{0, n+1\}$ and $A = \{(j,k) : j \in V \setminus \{n+1\},\ k \in V \setminus \{0\},\ j \neq k\}$, where 0 and $n+1$ represent origin and destination dummy nodes, respectively. Let σ_{jk} for $(j,k) \in A$, represent the setup times between jobs j and k, assuming value $s_{i(k)}$ (where $s_{i(k)}$ is the setup related to the family $i(k)$ of job k) if $i(j) \neq i(k)$, and 0 otherwise. The variables used by the OC model are: (i) a binary decision variable x_{jk}, assuming value 1 if job j is scheduled before job k, 0 otherwise, for all $(j,k) \in A$; (ii) a non-negative continuous variable C_j representing the completion time of job j; and (iii) a continuous variable τ_{jk} that represents the starting time of job k if scheduled immediately after job j, for all $(j,k) \in A$. The OC formulation is then:

$$(OC) \quad \min \sum_{j \in J} w_j\, C_j \tag{1}$$

$$s.t. \sum_{k \in V \setminus \{0\}} x_{jk} = 1 \qquad j \in J \tag{2}$$

$$\sum_{k \in V \setminus \{n+1\}} x_{kj} = 1 \qquad j \in J \tag{3}$$

$$\sum_{j \in V \setminus \{n+1\}} x_{0j} = m \tag{4}$$

$$\sum_{j \in V \setminus \{0\}} x_{j,n+1} = m \tag{5}$$

$$\sum_{k \in V \setminus \{n+1\}} (\tau_{kj} + \sigma_{kj} x_{kj}) + p_j = \sum_{k \in V \setminus \{0\}} \tau_{jk} \qquad j \in V \tag{6}$$

$$C_j = \sum_{k \in V \setminus \{0\}} \tau_{jk} \qquad j \in V \tag{7}$$

$$x_{jk} \in \{0,1\} \qquad (j,k) \in A \tag{8}$$

$$0 \leq \tau_{jk} \leq T x_{jk} \qquad (j,k) \in A \tag{9}$$

The objective function (1) seeks the minimum total weighted completion time. Constraints (2)–(5) are flow constraints ensuring that all jobs are processed exactly once and that m

machines are used. Constraints (6) and (7) define an ordered sequence of finishing and starting times of the jobs and their completion times. Constraints (8) define the domain of the x_{jk} variables, and constraints (9) impose the bounds of variables τ_{jk} and their relation with variables x_{jk}. Note that the model contains a polynomial number of variables $O(n^2)$ and constraints $O(n)$. Note also that the C_j variables are simply used as support variables as they could be replaced by the summation in (7).

Regarding Example 1, the OC optimal solution is: $C_1 = 11$, $C_2 = 7$, $C_3 = 5$, $C_4 = 12$, $C_5 = 7$, $C_6 = 20$; $x_{05} = x_{54} = x_{47} = x_{03} = x_{32} = x_{21} = x_{16} = x_{67} = 1$; $\tau_{05} = 0, \tau_{54} = 7, \tau_{47} = 12, \tau_{03} = 0, \tau_{32} = 5, \tau_{21} = 7, \tau_{16} = 11, \tau_{67} = 20$ (and all other variables take the value 0).

3.2. Arc-flow formulations

AF formulations represent combinatorial optimization problems by means of flows on a capacitated network. Regarding scheduling problems, these flows can be easily associated with schedules. AF formulations were successfully applied to bin packing and cutting stock problems (see, e.g., Valério de Carvalho 1999; Delorme et al. 2016) and vehicle routing problems (Macedo et al. 2011). Very recently, good results were also obtained in the scheduling field, but without setup times, by Kramer et al. (2018) and Mrad and Souayah (2018). To the best of our knowledge, this is the first time AF are used to model a scheduling problem with setup times. We propose three different ways for achieving this.

3.2.1. Family layered arc-flow formulation

Our first AF models the m sequences on the machines as paths on a network composed of a source node, a sink node and intermediate nodes distributed on f layers, one for each family. The nodes are connected by arcs on a set composed by: job arcs, that model the processing of a job; setup arcs, that model the transition from a family to another; and loss arcs, that simply lead to the sink node. Each path starts at the source node and finishes at the sink node.

Formally, for this AF model, that we call *family layered arc flow* (AF_{fl}), we are given a direct acyclic multi-graph $G = (\overline{N}, A)$. Each node in $\overline{N}$ is defined by a pair (i, q), where $i \in F \cup \{0\}$ either takes the index of a family or the dummy index 0, and q stands for the time. Instead of simply using the values of q in $0 \leq q \leq T$, we consider a reduced set by using $q \in N$, where $N \subseteq \{0, \ldots, T\}$ represents the set of *normal patterns*, i.e., the set of all feasible combinations of job processing times and family setup times up to T (we refer the reader to Côté and Iori 2018 for a formal definition of normal patterns). The set of nodes is partitioned in such a way that $\overline{N} = \cup_{i \in F} \overline{N}_i \cup \{(0,0), (0,T)\}$, with $\overline{N}_i = \{(i, q) : q \in N\}$ representing all possible starting and ending times of jobs in layer i, for all $i \in F$, and $(0, 0)$

and $(0, T)$ representing source and sink nodes, respectively. The set of arcs is partitioned as $A = A' \cup S \cup L \cup I$. The set $A' = \{(i, j, q, r) : i \in F; j \in J_i; q, r \in N; r = q + p_j\}$ contains the job arcs from node (i, q) to node (i, r), representing the fact that job j belonging to family i is processed during the time interval $[q, r)$. Note that the arcs in A' remain in the same layer i. Differently from A', the set $S = \{(i, h, q, r) : i, h \in F; i \neq h; q, r \in N; r = q + s_h\}$ is the set of setup arcs from node (i, q) to node (h, r), that represent a transition between the processing of two jobs from different families. In other words, an arc (i, h, q, r) states a change from family i to family h starting at time q, lasting s_h and consequently finishing at $r = q + s_h$. The set $L = \{(i, 0, q, T) : i \in F; q \in N\}$ models the loss arcs that link node (i, q) to the sink node $(0, T)$. Finally, the set $D = \{(0, i, 0, s_i) : i \in F\}$ contains the arcs from the origin node $(0, 0)$ to node (i, s_i), thus modeling the initial setup times and meaning how many times each family has the first job on the machines.

Our AF_{fl} model is based on the following variables: (i) binary variable x_{ijqr} assuming the value 1 if job arc $(i, j, q, r) \in A'$ is taken, 0 otherwise; (ii) an integer variable s_{ihqr} representing the number of times setup arc $(i, h, q, r) \in S$ is chosen; (iii) a continuous variable l_{iq} indicating how many loss arcs $(i, 0, q, T) \in L$ are selected; and (iv) a continuous variable d_i giving the number of times dummy arc $(0, i, 0, s_i) \in D$ is used. The $P|s_i| \sum w_j C_j$ can be modeled as an AF_{fl} as follows:

$$(AF_{fl}) \quad \min \sum_{(i,j,q,r)\in A'} w_j \, r \, x_{ijqr} \tag{10}$$

$$s.t. \sum_{(i,j,q,r)\in A'} x_{ijqr} \geq 1 \quad i \in F, j \in J_i \tag{11}$$

$$\sum_{(0,i,0,s_i)\in D} d_i = m \tag{12}$$

$$\sum_{(i,0,q,T)\in L} l_{iq} = m \tag{13}$$

$$\sum_{(ijpq)\in A'} x_{ijpq} - \sum_{(ijqr)\in A'} x_{ijqr} + \sum_{\substack{(ihqr)\in S \\ h\in F\setminus\{i\}}} s_{ihqr} - \sum_{\substack{(hipq)\in S \\ h\in F\setminus\{i\}}} s_{hipq} + l_{iq} = \begin{cases} d_i, \text{if } q = s_i & \\ 0, \text{otherwise} & \end{cases} \quad \begin{matrix} i\in F, \\ q\in\{0,\ldots,T-1\} \end{matrix} \tag{14}$$

$$x_{ijqr} \in \{0, 1\} \quad (i, j, q, r) \in A' \tag{15}$$

$$s_{ihqr} \in \{0, \ldots, |J_h|\} \quad (i, h, q, r) \in S \tag{16}$$

$$0 \leq d_i \leq m \quad (0, i, 0, s_i) \in D \quad (17)$$

$$0 \leq l_{iq} \leq m \quad (i, 0, q, T) \in L \quad (18)$$

Constraints (11) guarantee that all jobs are scheduled at least once; constraints (12) and (13) ensure that m machines are used. Constraints (14) impose flow conservation and thus ensure that only feasible schedules are obtained. The domains of the variables are defined by constraints (15)–(18). The AF_{fl} formulation has a pseudo-polynomial number of variables $O(\ (n + f^2)T\ )$ and constraints $O(fT)$.

Figure 2 shows an optimal solution of Example 1 by AF_{fl} formulation (10)–(18). The depicted solution contains two paths, each representing the schedule of a machine. The first path starts with an arc from the origin $(0, 0)$ to node $(1, 2)$, then all jobs from family 1 are sequenced and, before sequencing job 6 from family 3 and closing the path with variable $l_{3,20}$, the setup arc $(1, 3, 11, 14) \in S$ representing the change from family 1 to family 3 is used. The second path concerns the sequencing of jobs from family 2, where d_2 models the initial setup time for the family.

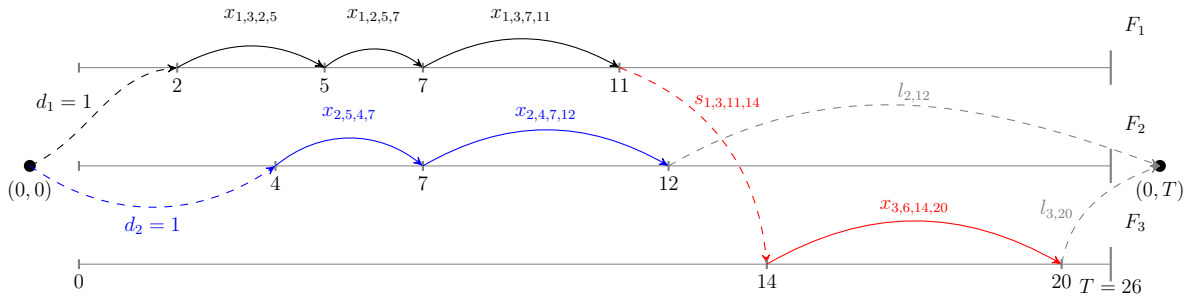

Figure 2: An AF_{fl} optimal solution of Example 1

3.2.2. Arc-flow formulation with alternating paths

Our second AF formulation, denoted as AF_{ap}, relies on the idea of shrinking the AF_{fl} layers into a unique layer. Our AF_{ap} still models the $P|s_i|\sum w_j C_j$ as the problem of finding m paths from source node 0 to sink node T in such a way that all jobs are scheduled. This time, however, Each path should alternate between (i) a job arc and (ii) either a setup, a dummy or a loss arc. To this aim, we consider a direct acyclic multigraph $G = (N, A)$ where the set of nodes $N \subseteq \{0, \ldots, T\}$ represents the set of normal patterns previously discussed. Set A contains job arcs $A' = \{(q, r, j);\ j \in J,\ q \in N,\ r = q + p_j\}$, representing the fact that job j starts at time $q \in N$ and finishes at $r = q + p_j$; setup arcs $S = \{(q, r, i);\ i \in F,\ q \in N,\ r = q + s_i\}$, representing a transition to family i starting in $q \in N$

and finishing in $r = q + s_i$; dummy arcs $D = \{(q, q, i);\ i \in F,\ q \in N\}$, representing a dummy setup of zero units of time between two jobs from the same family; and loss arcs $L = \{(q, T);\ q \in N \setminus \{0, T\}\}$, leading the path to the sink node T. In addition, let us define sets A'_j, $\delta^+_{(q)}$ and $\delta^-_{(q)}$ as the sets of job arcs of $j \in J$, the set of all arcs starting at time $q \in N$, and the set of all arcs ending at time $q \in N$, respectively.

The AF_{ap} formulation makes use of four sets of variables, namely: (i) a binary variable x_{qrj} taking value 1 if arc $(q, r, j) \in A'$ is selected, 0 otherwise; (ii) an integer variable s_{qri} indicating how many setup arcs $(q, r, i) \in S$ are selected; (iii) a continuous variable d_{qqi} for each dummy arc $(q, q, i) \in D$, giving the number of times a dummy arc from q to q of family i is selected; and (iv) a continuous variable l_{qT} representing the number of loss arcs $(q, T) \in L$ that are taken. Then, the $P|s_i|\sum w_j C_j$ can be modeled as:

$$(AF_{ap}) \quad \min \sum_{(q,r,j)\in A'} w_j\, r\, x_{qrj} \tag{19}$$

$$s.t. \sum_{(q,r,j)\in A'_j} x_{qrj} \geq 1 \qquad j \in J \tag{20}$$

$$\sum_{(0,r,i)\in\delta^+_{(0)}} s_{0ri} = m \tag{21}$$

$$\sum_{(q,T)\in L} l_{qT} + \sum_{(q,T,j)\in\delta^-_{(T)}} x_{qTj} = m \tag{22}$$

$$\sum_{(p,q,j)\in\delta^-_{(q)}} x_{pqj} = \sum_{(q,r,i)\in\delta^+_{(q)}} s_{qri} + \sum_{(q,q,i)\in D} d_{qqi} + l_{qT} \qquad q \in N \setminus \{0, T\} \tag{23}$$

$$d_{qqi} \leq \sum_{(p,q,j)\in\delta^-_{(q)}: j\in J_i} x_{pqj} \qquad q \in N \setminus \{0, T\}, i \in F \tag{24}$$

$$d_{qqi} \leq \sum_{(q,r,j)\in\delta^+(q): j\in J_i} x_{qrj} \qquad q \in N \setminus \{0, T\}, i \in F \tag{25}$$

$$\sum_{(q,r,j)\in\delta^+(q): j\in J_i} x_{qrj} = s_{pqi} + d_{qqi} \qquad q \in N \setminus \{0, T\}, i \in F \tag{26}$$

$$x_{qrj} \in \{0, 1\} \qquad (q, r, j) \in A' \tag{27}$$

$$s_{qri} \in \{0, \ldots, \max\{m, |J_i|\}\} \qquad (q, r, i) \in S \tag{28}$$

$$0 \leq d_{qqi} \leq \max\{m, |J_i|\} \qquad (q, q, i) \in D \tag{29}$$

$$0 \leq l_{qT} \leq m \qquad (q, T) \in L \tag{30}$$

Constraints (20) guarantee that all jobs are scheduled at least once; constraints (21)–(22)

force m machines to be used, starting with a setup arc and ending with either a loss or a job arc. Constraints (23)–(26) impose the flow conservation (in the way described in detail below), while in constraints (27)–(30) the domains of the variables are defined. AF_{ap} formulation (19)–(30) contains $O((n+f)T)$ variables and $O(fT)$ constraints. For the sake of clarity and ease of comprehension of the flow conservation constraints, Figures 3 and 4 illustrate constraints (23) and (26), respectively. From Figure 3, it is possible to note that constraints (23) state that every time a job arc finishes at time q, it must be followed by either a setup, a dummy or a loss arc starting at q. Following the same line of reasoning constraints (26), depicted in Figure 4, model the fact that every time a setup or a dummy arc of family i is used, it should be necessarily be followed by a job arc associated with this family .

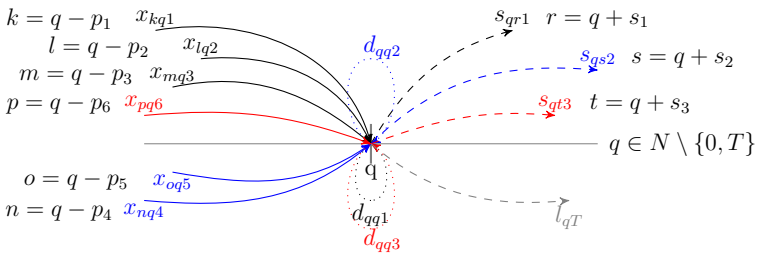

Figure 3: Illustration of constraints (23)

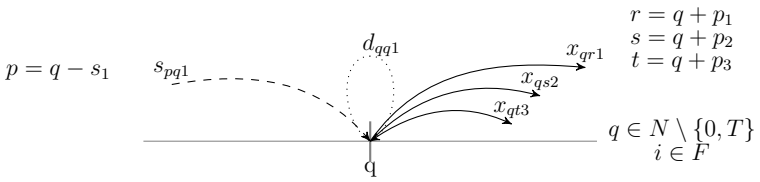

Figure 4: Illustration of constraints (26)

Concerning Example 1, an AF_{ap} optimal solution is shown in Figure 5. It is possible to identify two paths (one for each machine) starting with setup arcs and finishing with loss arcs. When two jobs belonging to the same family are scheduled consecutively one after the other, a dummy arc is present between them (this is the case for $d_{5,5,1}$, $d_{7,7,2}$ and $d_{9,9,1}$). If instead two jobs from different families are sequenced one after the other, then a setup ar is needed (this is the case of $(s_{11,14,3})$ that models the change from family 1 to family 3).

In addition, we devise some valid constraints that can be used to strengthen the linear relaxation of AF_{ap} and reduce the size of the enumeration tree. These constraints rely

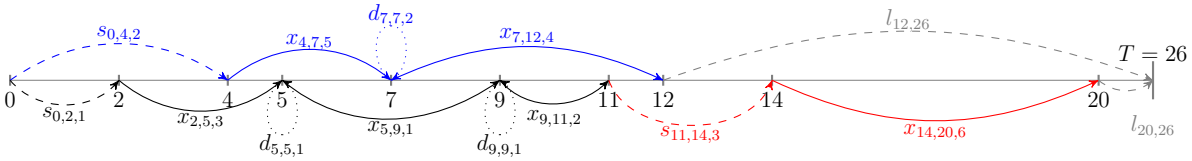

Figure 5: An AF_{ap} optimal solution ($z^* = 148$) of Example 1

on some basic observations about the problem and the AF_{ap} formulation itself and are as follows:

$$\sum_{(q,q,i)\in A} d_{qqi} + \sum_{(q,r,i)\in A} s_{qri} = |J_i| \qquad i \in F \tag{31}$$

$$\sum_{(q,r,i)\in A} s_{qri} \geq 1 \qquad i \in F \tag{32}$$

$$\sum_{(q,q,i)\in A} d_{qqi} \leq |J_i| - 1 \qquad i \in F \tag{33}$$

They state that, for each family $i \in F$, exactly $|J_i|$ setup + dummy arcs should be taken (31), at least one of them should be a setup arc (32), and no more than $|J_i| - 1$ of them can be dummy arcs (33). By considering these constraints alongside with AF_{ap} formulation (19)–(30), we obtain formulation (19)-(33), referred to as AF^+_{ap} in the following.

3.2.3. Arc-flow formulation with doubled jobs

Our third AF formulation, called *AF with doubled jobs* (AF_{dj}), still seeks for m paths on a capacitated network, but is supported by the idea of creating additional "long" job arcs that model both setup and processing of a job. The AF_{dj} formulation makes use of a direct acyclic multigraph $G = (N, A)$. The set of nodes N has the same meaning than the set of nodes in Section 3.2.2, whereas the set of arcs is now partitioned as $A = A' \cup A'' \cup L$. Again, sets A' and L have the same meaning as the sets of job and loss arcs in Section 3.2.2, while $A'' = \{(q, r, j);\ j \in J;\ q \in N;\ r = q + p_j + s_{i(j)}\}$ is the set of long job arcs. Each long job arc has a binary variable x'_{qrj} associated with, that assumes the value 1 if arc $(q, r, j) \in A''$ is taken, 0 otherwise. Similarly to AF_{ap}, sets $\delta^+_{(q)}$ and $\delta^-_{(q)}$ are the sets of all arcs (job and long job ones) starting at time $q \in N$, and of all arcs ending at time $q \in N$, respectively. The $P|s_i|\sum w_jC_j$ can be then modeled as:

$$(AF_{dj}) \quad \min \sum_{(q,r,j)\in A'} w_j\, r\, x_{qrj} + \sum_{(q,r,j)\in A''} w_j\, r\, x'_{qrj} \tag{34}$$

$$s.t. \sum_{(q,r,j)\in A'} x_{qrj} + \sum_{(q,r,j)\in A''} x'_{qrj} \geq 1 \qquad j \in J \tag{35}$$

$$\sum_{(0,q,j)\in A''} x'_{0qj} = m \tag{36}$$

$$\sum_{(q,T)\in L} l_{qT} + \sum_{(q,T,j)\in \delta^-_{(q)}} (x_{qTj} + x'_{qTj}) = m \tag{37}$$

$$\sum_{(q,r,j)\in \delta^+_{(q)}} (x_{qrj} + x'_{qrj}) + l_{qT} - \sum_{(p,q,j)\in \delta^-_{(q)}} (x_{pqj} + x'_{pqj}) = 0 \qquad q \in N \setminus \{0,T\} \tag{38}$$

$$\sum_{\substack{(p,q,j)\in \delta^-_{(q)} \\ j\in J_i}} (x_{pqj} + x'_{pqj}) \geq \sum_{\substack{(q,r,j)\in \delta^+_{(q)} \\ j\in J_i}} x_{qrj} \qquad i \in F : s_i > 0, q \in N \tag{39}$$

$$\sum_{(q,r,j)\in A''} x'_{qrj} \geq \max\{f, m\} \tag{40}$$

$$x_{qrj} \in \{0,1\} \qquad (q,r,j) \in A' \tag{41}$$

$$x'_{qrj} \in \{0,1\} \qquad (q,r,j) \in A'' \tag{42}$$

$$0 \leq l_{qT} \leq m \qquad (q,T) \in L \tag{43}$$

Constraints (35) require all jobs to be scheduled; constraints (36)-(37) force the m machines to be used. Constraints (38)-(39) impose flow conservation, constraint (40) states the minimum number of long arcs to be included in a solution. The domains of the variables are defined by constraints (41)-(43). AF_{dj} also contains a pseudo-polynomial number of variables $O(nT)$ and constraints $O(fT)$. An AF_{dj} optimal solution for Example 1 is presented in Figure 6, where $x_{5,9,1}$, $x_{7,12,4}$ and $x_{9,11,2}$ are job arcs; $x'_{0,7,5}$, $x'_{0,5,3}$ and $x'_{11,20,6}$ are long job arcs; and $l_{12,26}$ and $l_{20,26}$ are loss arcs. Note that the long arc $x'_{0,7,5}$ models the setup time to family $i(5) = 2$ ($s_{i(5)} = 4$) plus the processing time of job 5 ($p_5 = 3$).

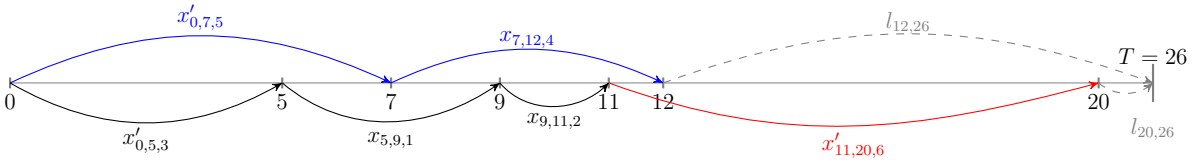

Figure 6: AF_{dj} optimal solution of Example 1

3.3. Set covering formulation

SC formulations are commonly used to model various combinatorial optimization problems, including but not limited to: bin packing (Valério de Carvalho, 1999), cutting stock

(Delorme et al., 2016), vehicle routing (Feillet, 2010) and scheduling (van den Akker et al., 1999). In our case, a set Ω containing all possible feasible single machine schedules, upper bounded by the time horizon T and respecting family and setup constraints, is given. Each schedule $\omega \in \Omega$ has a cost c_ω. Furthermore, let $a_{j\omega}$ be a coefficient that takes the value 1 if job $j \in J$ is present in schedule ω, 0 otherwise, and let x_ω be a binary variable assuming the value 1 if schedule ω is taken, 0 otherwise. The SC formulation is:

$$(SC) \quad \min \sum_{\omega \in \Omega} c_\omega \, x_\omega \tag{44}$$

$$s.t. \sum_{\omega \in \Omega} a_{j\omega} x_\omega \geq 1 \qquad j \in J \tag{45}$$

$$\sum_{\omega \in \Omega} x_\omega = m \tag{46}$$

$$x_\omega \in \{0, 1\} \qquad \omega \in \Omega \tag{47}$$

The objective function (44) aims at a solution of minimum cost. Constraints (45) ensure that all job are covered whereas constraint (46) impose the use of m schedules and constraints (47) define the domain of the variables.

The SC formulation (44)-(47) contains an exponential number of variables. Hence, explicit enumerating all of them is unpractical. Therefore, we propose a *branch-and-price* (B&P) algorithm to solve it. In general words, a B&P algorithm is a B&B in which each node of the tree is solved by means of a *column generation* (CG) approach. The CG method basically decomposes the model into a *master problem*(MP), which is initialized with a small subset of columns of Ω, and one or more subproblems, called pricing problems. To find promising columns, the CG algorithm benefits from the *linear program* (LP) dual information, given by the optimal solution of the continuous relaxations of the MP. The columns obtained by the resolution of the pricing problems are added on the fly to the MP. Thus, master and pricing problems are solved iteratively until no more attractive columns exist. In this way, it is possible to solve the SC formulation without the need of explicitly enumerating all columns. A detailed explanation of CG algorithm can be found in, e.g., Lübbecke and Desrosiers (2005).

In our B&P algorithm, the CG pricing subproblem consists of finding columns in Ω, i.e, feasible single machine schedules in the time interval $[0, T]$, that have negative reduced costs. To this aim, the first option would consist in choosing as subproblem the $\mathcal{NP}$-hard *elementary shortest path problem with resource constraints* (ESPPRC). As finding columns with the ESPPRC is a difficult task, we opted instead, on obtaining columns in a larger

set $\Omega' \supseteq \Omega$, by solving a relaxed version of the ESPPRC where a job can be included more than once in a column, but not tow or more times consecutively one after the other. The relaxed subproblem is defined as the problem of finding paths with negative reduced costs on a network $G = (N', A)$ where $N' = \{(j, q) : j \in J; s_{i(j)} \leq q \leq T - p_j\} \cup (0, T)$, is the set of nodes, with $(0, T)$ representing the sink node, and $A = \{(j, q, k, r) : (j, q), (k, r) \in N'; r = q + p_j, \text{ if } i(j) = i(k), r = q + p_j + s_{i(k)}, \text{ otherwise}\}$ is the set arcs. This problem can be solved efficiently in pseudo-polynomial time by a modified version of the DP algorithm presented in Pessoa et al. (2010) for the $P||\sum w_j T_j$. The DP algorithm that we use is shown in Algorithm 1.

Algorithm 1 DP

```
1:  procedure DP(G(N', A))
2:      initialize F(j, q) ← 0, if q = s_{i(j)};    F(j, q) ← +∞, otherwise; ∀(j, q) ∈ N;
3:      for q ∈ {0, ..., T} do
4:          for j, k ∈ J|j ≠ k and (j, q) ∈ N do
5:              c̄_j ← w_j(q + p_j) − λ_j;    r ← +∞;
6:              if i(j) ≠ i(k) then r ← q + p_j + s_{i(k)};
7:              else if k > j then r ← q + p_j;
8:              if (k, r + p_k) ∈ N and F(j, q) + c̄_j < F(k, r) then
9:                  F(k, r) ← F(j, q) + c̄_j;    F(0, T) ← min{F(0, T), F(k, r)}
10:     return F(0, T) − λ_0
```

In Algorithm 1, λ_0 and λ_j, $j \in J$, are the dual variables associated with constraint (46) and constraints (45), respectively. State $F(j, q)$ represents the reduced cost of a path that finishes with the processing of job j starting at time q.

Our B&P is initialized with the set of columns corresponding to the WSPT heuristic solution and the set of columns in the solution obtained by the metaheuristic by Kramer and Subramanian (2017), which we invoke before executing our B&P. The WSPT heuristic solution is obtained by iteratively sequencing the jobs (previously sorted according the WSPT order) on the current least-loaded machine, without considering the setup times. Once all jobs are sequenced, the setup times are included, thus obtaining a feasible solution for the $P|s_i|\sum w_j C_j$. Then, at each iteration of the CG algorithm, the column corresponding to the pricing solution with most negative reduced cost, given by Algorithm 1, is introduced in the MP. If the node LP solution is not integer and its lower bound is smaller than the overall lower bound, then branching is performed using the same branching rule as in van den Akker et al. (1999). The rule first identifies a job j satisfying $\sum_{\omega \in \Omega^*} C_j(\omega) x^*_\omega > \min\{C_j(\omega) : x^*_\omega > 0\}$, where: Ω^* is the set of columns in the optimal

solution of the MP; x^*_ω is the value of variable x_ω in this optimal solution; and $C_j(\omega)$ is the completion time of job j in column ω. It then creates two nodes, one by limiting $C_j \leq \min\{C_j(\omega) : x^*_\omega > 0\}$ and the other by imposing $C_j \geq \min\{C_j(\omega) : x^*_\omega > 0\} + 1$. In practice, branching is obtained by imposing new release dates and deadlines on the jobs at each node. Then, the nodes are explored using the best bound strategy.

4. Computational experiments

In this section, we present the computational experiments carried out to assess the performance of the methods proposed in Section 3. The models were code in C++ and solved using Gurobi Optimizer 7.0. The tests were executed by using a single thread on a computer equipped with a quad-core Intel Xeon E5530 2.40GHz processor, 20GB of RAM and Ubuntu 14.04.5 LTS operating system. In the experiments, all our proposed methods received an initial upper bound as cutoff value. This cutoff was obtained by the application of the *iterated local search* (ILS) metaheuristic of Kramer and Subramanian (2017) originally devise to deal with a large class of scheduling problems. For each instance, we ran the ILS with a time limit of $n/30$ seconds. In Section 4.1, we discuss the considered benchmark instances from the literature. The computational results are presented in Section 4.2.

4.1. Benchmark instances

In our experiments, we considered three sets of benchmark instances. The first and the second were originally proposed by Dunstall and Wirth (2005a) and Dunstall and Wirth (2005b), respectively. The first set is composed by 4800 instances, with $n \in \{10, 15, 20, 30\}$, $m \in \{2, 3, 5\}$ and $f \in \{1, 3, 5, 8\}$. The job processing times and weights were randomly drawn from the intervals $[p_{\min}, p_{\max}] = [1, 100]$ and $[w_{\min}, w_{\max}] = [1, 10]$, respectively. The family setup times were randomly generated in the range $[0, 50]$. For each combination of (n, m, f), 100 instances were created. The second set is composed by large sized instances with $n \in \{40, 80\}$, $m \in \{2, 3, 5\}$ and $f \in \{1, 3, 5, 8, 12, 16\}$. The processing times and weights of the jobs were created as in the previous set, but the setup times are now random integer numbers in the range $[0, s_{\max}]$, with $s_{\max} \in \{50, 100\}$. For each combination of $(n, m, f, s_{\max})$, 100 instances were created, resulting in a total of 7200 benchmark instances.

The last set of benchmark instances was proposed by Liao et al. (2012). This set is dived into small and large sized instances. Small sized instances have $n \in \{15, 20, 25, 30\}$, $m \in \{3, 4, 5\}$ and $f \in \{3, 5, 8\}$, while the large sized ones have $n \in \{40, 60, 80\}$, $m \in \{3, 6, 9\}$ and $f \in \{5, 8, 12, 16\}$. Processing times, weights and setup times were generated as in

Dunstall and Wirth (2005a). For each combination of (n, m, f), 100 instances were created, for a total of 7200 instances.

The three sets of instances from the literature used in this work are available for download at `http://www.or.unimore.it/site/home/online-resources.html`.

4.2. Computational results

In this section, we evaluate the performance of the proposed mathematical models, namely, *OC* (1)-(9), AF_{fl} (10)-(18), AF_{ap} (19)-(30), AF_{ap}^{+} (19)-(33), AF_{dj} (34)-(43) and *SC* (44)-(47).

4.2.1. Results on benchmark set 1

Concerning the experiments on benchmark set 1, a time limit of 600 seconds has been imposed on each execution. First, we provide in Tables 2 and 3 a comparative analysis among our proposed formulations. The tables report, for each method and group of 100 instances, the number of instances solved to the proven optimality, *#opt*, the average final gap, *gap(%)* and the average gap between the *linear programming* (LP) relaxation lower bound and the best upper bound, $gap_{lp}(\%)$.

It can be noticed that the *OC* performance is very poor, as the model fails in solving most of the instances with $n > 15$. This can be explained, by its weak LP relaxation, whose gap is slightly above 50% on average. On the contrary, the continuous relaxations of the flow based models AF_{fl}, AF_{ap}, AF_{ap}^{+} and AF_{dj} are much stronger, specially AF_{dj}, whose LP bound is around 1% for most of the cases. From these tables, it can also be noticed the positive impact of the proposed strengthening constraints (31)-(33), as indeed AF_{ap}^{+} produces LP bounds around 1% better than the ones of AF_{ap}, on average. Regarding the number of problems solved to the proven optimality, AF_{dj} and *SC* clearly outperform all other formulations. Both methods managed to solve all instances with up to 20 jobs. Concerning the problems involving 25 jobs, AF_{dj} solved to the proven optimality all of them with 5 machines, but missed 4 with 3 machines and 66 with 2 machines. The *SC* model, in turn, solved all 25 jobs instances but one.

We now compare, in Tables 4 and 5, our best methods, AF_{dj} and *SC*, with the ones by Dunstall and Wirth (2005a). In their work, the authors proposed two B&B methods, both based on a list-scheduling approach where partial schedules are built at each node of the tree, but differing in the branching scheme. The first method is based on a *best-processor* (BP) strategy, whereas the second one on a *least-loaded-processor* (LLP) strategy. As stopping criterion, the authors imposed a limit of 10 million for the number of nodes created during the execution of their B&B algorithms. Results have not been reported

Table 2: Results of the proposed formulations on instances from set 1 with $n \in \{10, 15\}$

n	f	m	OC			AF_{fl}			AF_{ap}			AF^{+}_{ap}			AF_{dj}			SC		
			#opt	gap (%)	gap_{lp} (%)	#opt	gap (%)	gap_{lp} (%)	#opt	gap (%)	gap_{lp} (%)	#opt	gap (%)	gap_{lp} (%)	#opt	gap (%)	gap_{lp} (%)	#opt	gap (%)	gap_{lp} (%)
10	1	2	92	0.7	57.5	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
		3	98	0.2	46.1	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
		5	100	0.0	29.6	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
	3	2	100	0.0	61.3	100	0.0	3.8	100	0.0	3.5	100	0.0	2.2	100	0.0	0.5	100	0.0	0.0
		3	100	0.0	49.6	100	0.0	1.3	100	0.0	1.1	100	0.0	0.6	100	0.0	0.3	100	0.0	0.2
		5	100	0.0	33.9	100	0.0	0.3	100	0.0	0.2	100	0.0	0.2	100	0.0	0.1	100	0.0	0.1
	5	2	100	0.0	60.8	100	0.0	7.3	100	0.0	5.1	100	0.0	1.7	100	0.0	0.5	100	0.0	0.0
		3	100	0.0	48.5	100	0.0	3.4	100	0.0	2.0	100	0.0	0.4	100	0.0	0.2	100	0.0	0.0
		5	100	0.0	31.9	100	0.0	0.8	100	0.0	0.3	100	0.0	0.1	100	0.0	0.1	100	0.0	0.0
	8	2	100	0.0	56.5	99	0.0	11.6	100	0.0	3.1	100	0.0	0.5	100	0.0	0.1	100	0.0	0.0
		3	100	0.0	42.6	100	0.0	6.5	100	0.0	1.3	100	0.0	0.1	100	0.0	0.0	100	0.0	0.0
		5	100	0.0	24.5	100	0.0	2.1	100	0.0	0.3	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
15	1	2	0	36.6	67.3	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
		3	0	22.8	57.6	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
		5	22	9.0	43.6	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
	3	2	0	33.7	69.6	99	0.0	4.2	99	0.0	4.1	99	0.0	3.6	100	0.0	1.0	100	0.0	0.0
		3	2	15.4	59.3	100	0.0	1.5	100	0.0	1.5	100	0.0	1.2	100	0.0	0.5	100	0.0	0.2
		5	73	1.3	45.1	100	0.0	0.5	100	0.0	0.4	100	0.0	0.4	100	0.0	0.3	100	0.0	0.3
	5	2	0	30.7	71.2	92	0.2	8.7	92	0.3	8.1	98	0.1	5.7	100	0.0	0.9	100	0.0	0.0
		3	1	11.8	60.7	100	0.0	4.3	100	0.0	3.9	100	0.0	2.3	100	0.0	0.4	100	0.0	0.0
		5	98	0.0	46.1	100	0.0	1.2	100	0.0	1.0	100	0.0	0.5	100	0.0	0.3	100	0.0	0.1
	8	2	0	29.8	70.5	21	4.2	12.5	83	0.8	8.5	97	0.1	4.8	100	0.0	0.8	100	0.0	0.0
		3	2	10.9	59.6	95	0.1	7.9	100	0.0	4.9	100	0.0	2.1	100	0.0	0.4	100	0.0	0.0
		5	100	0.0	43.8	100	0.0	3.0	100	0.0	1.5	100	0.0	0.3	100	0.0	0.1	100	0.0	0.1
Avg.			1488	8.5	51.5	2306	0.2	3.4	2374	0.0	2.1	2394	0.0	1.1	2400	0.0	0.3	2400	0.0	0.0

by the authors for the groups of 100 instances, defined by (n, f, m), when the stopping criterion have been reached for at least four instances.

According to the single thread results presented in `https://www.cpubenchmark.net/`, the Intel Xeon E5530 2.40 GHz processor used in our experiments is about 2.5 times faster than an Intel Pentium III 1.40GHz processor, which is similar to the one used by Dunstall and Wirth (2005a) in their experiments. Thus, in Tables 4 and 5, the reported execution times by Dunstall and Wirth (2005a) have been divided by 2.5 to facilitate the comparison. For each method and group of instances in a line, the tables report the number of instances solved to the proven optimality *#opt*, the average number of nodes enumerated *#nodes* and the average execution time elapsed in seconds *t(s)*.

With regard to the results in Table 4, all methods were able to solve very quickly all considered instances with 10 and 15 jobs. The main difference among the methods concerns the number of explored nodes required to prove the optimality. While AF_{dj} and SC prove very often the optimality at the root node, the B&B methods need a higher effort, exploring

Table 3: Results of the proposed formulations on instances from set 1 with $n \in \{20, 25\}$

n	f	m	OC			AF_{fl}			AF_{ap}			AF^{+}_{ap}			AF_{dj}			SC		
			#opt	gap (%)	gap_{lp} (%)	#opt	gap (%)	gap_{lp} (%)	#opt	gap (%)	gap_{lp} (%)	#opt	gap (%)	gap_{lp} (%)	#opt	gap (%)	gap_{lp} (%)	#opt	gap (%)	gap_{lp} (%)
20	1	2	0	49.6	73.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
		3	0	36.0	63.9	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
		5	0	19.9	50.3	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
	3	2	0	50.6	75.2	82	0.4	4.6	64	1.0	4.6	73	0.6	4.4	100	0.0	1.4	100	0.0	0.0
		3	0	34.6	66.0	100	0.0	1.5	100	0.0	1.5	100	0.0	1.4	100	0.0	0.5	100	0.0	0.1
		5	0	16.9	53.0	100	0.0	0.6	100	0.0	0.6	100	0.0	0.5	100	0.0	0.4	100	0.0	0.3
	5	2	0	49.8	76.4	15	3.3	8.9	15	4.3	8.7	23	3.4	7.1	100	0.0	1.7	100	0.0	0.0
		3	0	33.2	67.2	96	0.1	4.8	94	0.1	4.7	97	0.0	3.6	100	0.0	0.9	100	0.0	0.1
		5	0	14.0	53.7	100	0.0	1.5	100	0.0	1.4	100	0.0	0.9	100	0.0	0.4	100	0.0	0.2
	8	2	0	48.4	77.3	2	8.0	13.4	9	6.6	11.5	17	4.0	8.3	100	0.0	1.4	100	0.0	0.0
		3	0	31.7	68.2	33	2.1	8.9	72	0.9	7.4	89	0.2	4.9	100	0.0	0.8	100	0.0	0.0
		5	0	11.1	54.1	100	0.0	3.7	100	0.0	2.8	100	0.0	1.5	100	0.0	0.4	100	0.0	0.1
25	1	2	0	57.7	77.7	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
		3	0	44.7	69.7	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
		5	0	28.2	57.5	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0	100	0.0	0.0
	3	2	0	59.1	78.9	34	2.0	4.7	19	3.0	4.7	18	3.0	4.7	75	0.5	1.8	100	0.0	0.0
		3	0	44.2	70.3	98	0.0	1.0	95	0.1	1.0	96	0.0	1.0	98	0.0	0.4	100	0.0	0.1
		5	0	26.6	58.4	99	0.0	0.7	100	0.0	0.7	100	0.0	0.7	100	0.0	0.6	99	0.0	0.6
	5	2	0	58.9	79.8	2	6.0	9.3	3	6.9	9.1	3	6.3	8.3	71	0.6	2.0	100	0.0	0.0
		3	0	44.0	71.6	59	1.1	5.2	51	1.7	5.1	53	1.4	4.5	98	0.0	1.1	100	0.0	0.0
		5	0	24.3	58.9	100	0.0	1.7	100	0.0	1.6	100	0.0	1.3	100	0.0	0.6	100	0.0	0.3
	8	2	0	58.1	79.2	0	9.7	12.3	0	9.1	11.6	0	7.0	9.3	88	0.2	1.6	100	0.0	0.0
		3	0	43.0	70.6	3	4.7	8.3	14	3.6	7.7	32	2.3	6.0	100	0.0	1.1	100	0.0	0.0
		5	0	22.3	56.8	92	0.1	3.4	98	0.0	3.1	100	0.0	2.1	100	0.0	0.4	100	0.0	0.1
Avg.			0	37.8	67.0	1715	1.6	3.9	1734	1.6	3.7	1801	1.2	2.9	2330	0.1	0.7	2399	0.0	0.1

a larger amount of nodes. The same trend is also observed in Table 5, but on a larger scale. The results in this table indicate that, when the instance size and the number of machines increase, the number of explored nodes by BP and LLP grows very rapidly. Regarding the instances with 20 jobs, it can be noticed that AF_{dj} and SC managed to solve all of them in reduced computational times, thus closing 3 open instances. Concerning the results on instances with 25 jobs, all 1200 instances have been solved either by AF_{dj} or SC. Overall, SC performed better than AF_{dj} on this group, as it managed to solve all instances but one.

4.2.2. Results on benchmark set 2

The second set of instances was proposed by Dunstall and Wirth (2005b). In their work, the authors developed heuristic algorithms and evaluated them by comparison with lower bounds from the literature. In Tables 6 and 7 we thus opted to compare the results of our best methods with the best lower and upper bounds of Dunstall and Wirth (2005b). Table 6 shows the results obtained for all 3600 instances with $n = 40$ within a time limit of 600 seconds, whereas Table 7 presents the aggregated results for a subset of 180 out of

Table 4: Comparison of AF_{dj} and SC with the B&B methods BP and LLP by Dunstall and Wirth (2005a) - instances from set 1 with $n \in \{10, 15\}$

n	f	m	AF_{dj}			SC			BP			LLP		
			#opt	#nodes	t(s)	#opt	#nodes	t(s)	#opt	#nodes	t(s)	#opt	#nodes	t(s)
10	1	2	100	0.0	0.3	100	0.0	<0.1	100	44.9	<0.1	100	44.5	<0.1
		3	100	0.0	0.2	100	0.0	<0.1	100	157.4	<0.1	100	154.6	<0.1
		5	100	0.0	0.1	100	0.0	<0.1	100	596.4	<0.1	100	598.3	<0.1
	3	2	100	0.0	0.6	100	0.1	<0.1	100	72.8	<0.1	100	39.7	<0.1
		3	100	0.0	0.3	100	0.9	<0.1	100	249.6	<0.1	100	109.0	<0.1
		5	100	0.0	0.1	100	0.6	<0.1	100	781.7	0.1	100	573.3	<0.1
	5	2	100	0.0	0.5	100	0.0	<0.1	100	70.1	<0.1	100	48.5	<0.1
		3	100	0.0	0.3	100	0.4	<0.1	100	240.0	<0.1	100	131.0	<0.1
		5	100	0.0	0.1	100	0.2	<0.1	100	741.3	0.1	100	534.7	0.1
	8	2	100	0.0	0.4	100	0.0	<0.1	100	41.0	<0.1	100	35.1	<0.1
		3	100	0.0	0.2	100	0.0	<0.1	100	125.0	<0.1	100	103.3	<0.1
		5	100	0.0	0.1	100	0.1	<0.1	100	327.8	<0.1	100	305.3	<0.1
15	1	2	100	0.0	0.7	100	0.0	<0.1	100	183.2	<0.1	100	184.3	<0.1
		3	100	0.0	0.4	100	0.0	<0.1	100	1869.9	0.1	100	1871.7	0.2
		5	100	0.0	0.3	100	0.1	<0.1	100	36607.2	2.7	100	36706.6	2.8
	3	2	100	0.4	3.6	100	0.2	0.2	100	947.5	0.1	100	254.6	<0.1
		3	100	0.5	1.3	100	3.4	0.1	100	7357.5	0.8	100	1196.4	0.2
		5	100	0.1	0.4	100	5.1	<0.1	100	84821.2	7.6	100	11036.4	1.3
	5	2	100	0.1	3.3	100	0.1	0.2	100	698.0	0.1	100	268.4	0.1
		3	100	0.0	1.0	100	0.5	0.1	100	6150.3	0.9	100	958.4	0.2
		5	100	0.1	0.4	100	1.8	<0.1	100	67421.3	7.4	100	10281.2	1.5
	8	2	100	0.3	2.7	100	0.2	0.2	100	555.5	0.1	100	328.2	0.1
		3	100	0.0	0.9	100	0.1	0.1	100	4257.6	0.7	100	1030.0	0.2
		5	100	0.0	0.3	100	1.2	<0.1	100	38362.7	5.6	100	11872.4	2.1
Avg.			2400	0.1	0.8	2400	0.6	<0.1	2400	10528.3	1.1	2400	3277.7	0.4

3600 instances with $n = 80$, solved within a larger time limit of 1 hour. In these tables, the best results by Dunstall and Wirth (2005b) are reported under the label $D\&W$. Columns *gap(%)* shows the average percentage gap for each method. Concerning the results from the literature, the gaps are calculated taking into consideration the best lower and upper bounds available. Finally, columns lb_{lp}, lb and ub under the tag *#best*, indicate how many times the referred method provided the best continuous relaxation (only for AF_{dj} and SC), and the best lower and upper bounds, respectively.

From Table 6, it can be noticed that AF_{dj} and SC are both able to improve most of the best lower and upper bounds from the literature. Indeed, all such values were improved or at least equaled by either AF_{dj} or SC. From a total of 3600 instances, AF_{dj} and SC solved to the proven optimality 1847 and 3395 of them, respectively. This result indicates the superiority of SC when compared to AF_{dj}. In part, this is due to the stronger LP relaxation provided by the SC formulation. Indeed, SC always provided the best LP bound. The performance of SC seems to worsen when the number of families decreases

Table 5: Comparison of AF_{dj} and SC with the B&B methods BP and LLP by Dunstall and Wirth (2005a) - instances from set 1 with $n \in \{20, 25\}$

n	f	m	AF_{dj}			SC			BP			LLP		
			#opt	#nodes	t(s)	#opt	#nodes	t(s)	#opt	#nodes	t(s)	#opt	#nodes	t(s)
20	1	2	100	0.0	1.1	100	0.0	0.1	100	922.5	0.1	100	918.6	0.1
		3	100	0.0	0.7	100	0.0	0.1	100	20210.8	2.3	100	20587.3	2.4
		5	100	0.0	0.4	100	0.0	<0.1	97	1140922.8	104.2	97	1141493.7	107.1
	3	2	100	78.8	25.6	100	0.6	0.8	100	9167.3	1.4	100	870.8	0.2
		3	100	17.8	5.5	100	8.2	0.8	100	197119.6	27.3	100	5257.7	1.1
		5	100	3.5	1.5	100	19.7	0.4				100	185532.4	28.4
	5	2	100	37.6	28.1	100	0.2	0.8	100	5760.0	1.4	100	1307.8	0.4
		3	100	16.9	5.7	100	4.4	0.6	100	201150.8	34.5	100	12691.3	2.8
		5	100	0.4	1.2	100	6.4	0.2				100	209972.9	45.4
	8	2	100	54.9	17.6	100	0.2	0.8	100	6239.1	1.6	100	2515.6	0.7
		3	100	0.4	4.2	100	0.4	0.4	100	54613.1	12.4	100	6737.8	2.1
		5	100	0.1	1.0	100	3.9	0.2				100	337673.7	67.2
25	1	2	100	0.0	2.0	100	0.0	0.4	100	3082.2	0.4	100	3100.8	0.4
		3	100	0.0	1.3	100	0.0	0.2	100	145909.0	17.0	100	149480.0	17.7
		5	100	0.0	0.7	100	0.0	0.1						
	3	2	75	1053.5	235.7	100	0.4	2.6	100	71794.3	12.9	100	4767.8	1.2
		3	98	118.7	22.3	100	32.1	10.5				100	16146.2	4.3
		5	100	80.3	7.2	99	191.1	15.6				97	1304194.4	276.8
	5	2	71	1473.1	251.5	100	0.2	2.5	100	65288.8	16.3	100	7006.9	2.2
		3	98	85.8	34.3	100	1.5	1.3				100	50762.7	18.5
		5	100	10.8	4.1	100	32.3	1.8				97	1016572.1	311.6
	8	2	88	826.0	136.2	100	0.2	2.6	100	70235.8	20.7	100	15391.0	4.7
		3	100	4.1	15.5	100	0.6	1.3				100	88857.1	28.1
		5	100	0.6	2.7	100	5.2	0.7						
Avg.			2330	161.0	33.6	2399	12.8	1.9	1397	140170.9	18.0	2191	208265.4	42.0

(but with $f > 1$) and the number of machines increases. In turn, AF_{dj} seems to perform better when the number of machines increases. Thus, one could affirm that these methods somehow complement each other. In general, both methods provided very low average gaps (around 1.1% for AF_{dj} and 0.1% for SC), within reasonable average execution times (around 5 minutes for AF_{dj} and less than a minute for SC). By considering together the best results of AF_{dj} and SC, we were able to solve to the proven optimality 3490 out of 3600 instances with $n = 40$.

The results on the instances with $n = 80$ are provided in Table 7. Again, it can be noticed that AF_{dj} and SC improve the best lower and upper bounds by Dunstall and Wirth (2005b). Regarding these instances, AF_{dj} solved 32 of them to the proven optimality (where 30 solved instances have a single family of jobs). SC, in turn, was able to prove the optimality of 135 out of 180 instances. In addition, SC provided all the best continuous bounds and most of the best final lower and upper bounds. On average, for the instances with 80 jobs, AF_{dj} provided a gap of 2.5% and SC a gap lower than 0.1%.

Table 6: Comparison of AF_{dj} and SC with the best results by Dunstall and Wirth (2005b) - instances from set 2 with $n = 40$

				D&W			*AF_{dj}*						*SC*					
n	$s_{\max}$	f	m	gap	#best		#opt	t(s)	gap	#best			#opt	t(s)	gap	#best		
				(%)	lb	ub			(%)	lb_{lp}	lb	ub			(%)	lb_{lp}	lb	ub
40	50	1	2	0.1	0	0	100	4.5	<0.1	98	100	100	100	4.9	<0.1	100	99	100
			3	0.3	0	0	100	3.0	<0.1	97	100	100	100	3.1	<0.1	100	100	100
			5	0.9	0	0	100	1.7	<0.1	99	100	100	100	1.5	<0.1	100	100	100
		3	2	3.2	0	15	17	527.9	1.3	0	17	99	99	49.3	<0.1	100	100	100
			3	4.7	0	6	50	364.0	0.4	2	50	99	79	162.6	<0.1	100	97	100
			5	4.7	0	1	81	180.9	0.1	7	90	100	55	290.8	0.2	100	65	97
		5	2	3.4	0	6	1	598.9	2.1	0	1	100	100	36.3	<0.1	100	100	100
			3	4.2	0	4	28	471.3	0.9	0	28	99	96	50.9	<0.1	100	100	100
			5	6.7	0	0	81	193.4	0.1	0	84	100	70	228.5	0.1	100	81	99
		8	2	3.5	0	2	0	tlim	2.4	0	0	98	100	35.0	<0.1	100	100	100
			3	3.8	0	2	20	522.9	1.3	0	20	97	100	16.3	<0.1	100	100	100
			5	5.2	0	1	94	87.3	<0.1	0	94	100	95	39.8	<0.1	100	97	100
		12	2	3.0	0	1	0	tlim	2.0	0	0	90	100	35.1	<0.1	100	100	100
			3	3.3	0	0	28	497.2	1.0	0	28	93	100	15.2	<0.1	100	100	100
			5	4.5	0	0	96	81.2	<0.1	0	96	100	100	13.9	<0.1	100	100	100
		16	2	2.6	0	0	10	580.9	1.4	0	10	86	100	31.7	<0.1	100	100	100
			3	2.9	0	0	63	353.0	0.5	0	63	83	100	14.3	<0.1	100	100	100
			5	3.8	0	0	99	47.9	<0.1	0	99	100	100	6.2	<0.1	100	100	100
40	100	1	2	0.1	0	0	100	6.8	<0.1	100	100	100	100	5.1	<0.1	100	100	100
			3	0.3	0	0	100	4.4	<0.1	100	100	100	100	3.7	<0.1	100	100	100
			5	0.7	0	0	100	2.4	<0.1	98	100	100	100	1.7	<0.1	100	100	100
		3	2	3.9	0	24	5	575.9	2.4	0	5	100	99	58.7	<0.1	100	100	100
			3	5.4	0	14	52	353.9	0.6	2	52	100	84	134.7	<0.1	100	98	100
			5	4.7	0	0	80	210.4	0.2	9	90	100	49	329.8	0.3	100	59	100
		5	2	3.8	0	19	0	tlim	3.6	0	0	100	100	45.0	<0.1	100	100	100
			3	4.2	0	8	22	504.9	1.5	0	22	100	99	34.2	<0.1	100	100	100
			5	6.6	0	2	82	193.7	0.2	1	82	100	73	213.1	<0.1	100	87	100
		8	2	3.4	0	4	0	tlim	4.1	0	0	98	100	47.7	<0.1	100	100	100
			3	3.9	0	2	6	580.6	2.3	0	6	99	100	19.7	<0.1	100	100	100
			5	5.8	0	4	95	124.3	<0.1	0	95	100	97	31.0	<0.1	100	99	100
		12	2	2.7	0	1	0	tlim	3.5	0	0	97	100	46.2	<0.1	100	100	100
			3	3.2	0	0	12	571.9	2.1	0	12	91	100	20.9	<0.1	100	100	100
			5	4.4	0	0	93	138.1	<0.1	0	93	100	100	7.4	<0.1	100	100	100
		16	2	2.1	0	0	5	587.4	2.6	0	5	97	100	44.3	<0.1	100	100	100
			3	2.5	0	0	33	506.9	1.3	0	33	88	100	18.5	<0.1	100	100	100
			5	3.6	0	0	94	107.9	<0.1	0	94	99	100	6.7	<0.1	100	100	100
Sum/Avg.				3.4	0	116	1847	332.9	1.1	613	1869	3513	3395	58.4	<0.1	3600	3482	3596

4.2.3. Results on benchmark set 3

This section present the computational results for the benchmark instances by Liao et al. (2012). This set was also used by Tseng and Lee (2017) to evaluate the performance of their metaheuristic. We found some small numerical inconsistencies in the results reported in both works, so we opted not to compare our results with theirs. Thus, we only performed a comparative analysis between our two best methods, AF_{dj} and SC. The instances of this

Table 7: Comparison of AF_{dj} and SC with the best results by Dunstall and Wirth (2005b) - instances from set 2 with $n = 80$

				Literature			AF_{dj}						SC					
n	$s_{\max}$	f	m	gap	#best		#opt	t(s)	gap	#best			#opt	t(s)	gap	#best		
				(%)	lb	ub			(%)	lb_{lp}	lb	ub			(%)	lb_{lp}	lb	ub
80	50	1	2	0.0	0	0	5	27.0	<0.1	5	5	4	5	1193.1	<0.1	5	5	5
			3	0.1	0	0	5	27.6	<0.1	5	5	5	5	857.5	<0.1	5	5	5
			5	0.3	0	0	5	12.6	<0.1	5	5	5	5	239.2	<0.1	5	5	5
		3	2	2.9	0	1	0	tlim	1.4	0	0	4	2	3358.8	<0.1	5	5	5
			3	4.5	0	0	0	tlim	0.8	0	0	4	2	2372.9	<0.1	5	5	5
			5	4.4	0	0	2	3396.1	0.5	0	2	5	1	3030.2	0.3	5	4	4
		5	2	3.3	0	0	0	tlim	2.3	0	0	3	3	2339.1	<0.1	5	5	5
			3	3.5	0	0	0	tlim	1.4	0	0	3	3	1910.2	<0.1	5	5	5
			5	5.7	0	0	0	tlim	0.8	0	1	4	1	2970.3	0.3	5	4	5
		8	2	4.0	0	0	0	tlim	4.0	0	0	1	4	2710.6	<0.1	5	5	5
			3	3.7	0	0	0	tlim	2.9	0	0	1	5	820.1	<0.1	5	5	5
			5	4.3	0	0	0	tlim	1.3	0	0	3	3	1726.8	<0.1	5	5	5
		12	2	4.1	0	0	0	tlim	4.3	0	0	0	5	2593.8	<0.1	5	5	5
			3	3.9	0	0	0	tlim	3.5	0	0	0	5	877.8	<0.1	5	5	5
			5	4.4	0	0	0	tlim	2.2	0	0	0	5	430.7	<0.1	5	5	5
		16	2	3.3	0	0	0	tlim	3.4	0	0	1	5	2138.4	<0.1	5	5	5
			3	3.4	0	0	0	tlim	2.8	0	0	0	5	1175.6	<0.1	5	5	5
			5	3.7	0	0	0	tlim	1.8	0	0	0	5	383.5	<0.1	5	5	5
80	100	1	2	0.0	0	0	5	43.8	<0.1	4	5	5	3	2883.8	<0.1	5	5	3
			3	0.1	0	0	5	32.8	<0.1	5	5	5	4	1820.9	<0.1	5	5	4
			5	0.3	0	0	5	16.8	<0.1	5	5	5	5	408.7	<0.1	5	5	5
		3	2	3.8	0	2	0	tlim	2.9	0	0	4	2	3171.9	<0.1	5	5	5
			3	6.4	0	0	0	tlim	2.1	0	0	2	3	2281.3	0.3	5	5	5
			5	6.0	0	0	0	tlim	0.8	0	0	4	2	2250.1	0.3	5	5	5
		5	2	3.0	0	1	0	tlim	3.3	0	0	5	4	2940.2	<0.1	5	5	5
			3	3.7	0	0	0	tlim	2.0	0	0	4	3	2111.7	<0.1	5	5	5
			5	6.1	0	0	0	tlim	0.8	0	0	5	2	2442.4	0.2	5	5	5
		8	2	4.5	0	0	0	tlim	6.3	0	0	1	3	3285.8	<0.1	5	5	5
			3	4.2	0	0	0	tlim	4.8	0	0	0	5	1056.2	<0.1	5	5	5
			5	4.3	0	0	0	tlim	1.7	0	0	3	4	1246.1	0.1	5	5	5
		12	2	4.2	0	0	0	tlim	6.4	0	0	2	4	3256.9	<0.1	5	5	5
			3	4.4	0	0	0	tlim	5.1	0	0	2	5	2017.4	<0.1	5	5	5
			5	5.1	0	0	0	tlim	3.0	0	0	1	5	998.1	<0.1	5	5	5
		16	2	3.4	0	0	0	tlim	6.8	0	0	2	2	3263.7	<0.1	5	5	5
			3	3.5	0	0	0	tlim	5.5	0	0	1	5	1525.3	<0.1	5	5	5
			5	3.7	0	0	0	tlim	3.5	0	0	0	5	369.5	<0.1	5	5	5
Sum/Avg.				3.5	0	4	32	2998.8	2.5	29	33	94	135	1901.6	<0.1	180	178	176

set with up to 40 jobs, namely, 4800 instances, have been considered and, for each run, a time limit of 600 seconds has been used. These results are detailed on Table 8, where, for each method and group containing 100 instances each, it is reported the number of instances solved to the proven optimality, *#opt*, the average final gap, *gap(%)*, the average gap between the linear relaxation and the best feasible solution, *gap(%)*, and the average computational time in seconds, *t(s)*. Both methods obtained very good results, with an

average gap lower than 0.1% and 4595 instances solved to proven optimality. AF_{dj} solved all instances with $n \leq 25$, while SC missed three of them. To obtain these results, AF_{dj} needed an average computational time of 42 seconds. SC, in turn, required around 36 seconds on average. In total, from the 4800 instances considered, only 81 of them are left open.

5. Conclusions

In this work, the problem of scheduling jobs on identical parallel machines with family setup times to minimize the total weighted completion time, $P|s_i|\sum w_jC_j$, has been tackled by exact methods. Five novel formulations have been proposed to solve the problem, namely, a one-commodity (OC), three arc-flow (AF_{fl}, AF_{ap} and AF_{dj}) and a set-covering (SC) formulation. Extensive computational experiments on benchmark instances from the literature have been performed and our results have been compared with the ones by the state-of-the-art methods. Among the proposed models, AF_{dj} and SC proved to have a better performance, being able to deal with instances containing up to 80 jobs in reasonable computational times. These good results can be explained, in part, by the strong continuous bounds provided by these models.

The $P|s_i|\sum_j w_jC_j$ demonstrated to be a very challenging problem, nevertheless the AF and SC formulations obtained relevant results and seem to be promising methods in this area. Therefore, our research is now directed on the application of AF and SC formulations to other machine scheduling problems involving different features, such as, e.g., release dates and unrelated parallel machines.

Acknowledgments

This research was partially funded by the CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico, Brazil, grant No. 234814/2014-4 and by University of Modena and Reggio Emilia, under grant FAR 2018 Analysis and optimization of healthcare and pharmaceutical logistic processes. We thank Simon Dunstall for providing us with the benchmark instances and the detailed results used in his research.

References

Allahverdi, A., 2015. The third comprehensive survey on scheduling problems with setup times/costs. European Journal of Operational Research 246 (2), 345–378.

Table 8: Results for instances from benchmark set 3 with up to 40 jobs

n	f	m	AF_{dj}				SC			
			#opt	gap (%)	gap_{lp} (%)	t(s)	#opt	gap (%)	gap_{lp} (%)	t(s)
15	3	3	100	0.0	0.6	1.2	100	0.0	0.2	0.1
		4	100	0.0	0.2	0.6	100	0.0	0.1	<0.1
		5	100	0.0	0.2	0.4	100	0.0	0.1	<0.1
	5	3	100	0.0	0.4	1.0	100	0.0	<0.1	0.1
		4	100	0.0	0.3	0.6	100	0.0	0.2	<0.1
		5	100	0.0	0.2	0.3	100	0.0	<0.1	<0.1
	8	3	100	0.0	0.4	0.9	100	0.0	<0.1	0.1
		4	100	0.0	0.2	0.5	100	0.0	<0.1	<0.1
		5	100	0.0	0.2	0.3	100	0.0	0.1	<0.1
20	3	3	100	0.0	0.7	5.3	100	0.0	0.2	1.0
		4	100	0.0	0.4	2.1	100	0.0	0.2	0.8
		5	100	0.0	0.3	1.1	100	0.0	0.1	0.2
	5	3	100	0.0	0.6	4.5	100	0.0	0.1	1.6
		4	100	0.0	0.5	2.4	100	0.0	0.3	0.4
		5	100	0.0	0.3	1.2	100	0.0	0.2	0.2
	8	3	100	0.0	0.6	4.1	100	0.0	0.2	1.1
		4	100	0.0	0.5	2.1	100	0.0	0.2	1.0
		5	100	0.0	0.3	1.0	100	0.0	0.1	0.3
25	3	3	100	0.0	0.7	23.0	100	0.0	0.2	6.0
		4	100	0.0	0.5	11.2	98	<0.1	0.3	23.9
		5	100	0.0	0.3	5.0	99	<0.1	0.2	11.6
	5	3	100	0.0	0.7	16.9	100	0.0	0.2	2.8
		4	100	0.0	0.5	5.7	100	0.0	0.2	2.3
		5	100	0.0	0.4	3.7	100	0.0	0.2	1.8
	8	3	100	0.0	0.7	15.4	100	0.0	0.2	7.5
		4	100	0.0	0.5	5.8	100	0.0	0.2	10.1
		5	100	0.0	0.4	3.2	100	0.0	0.2	2.3
30	3	3	95	<0.1	0.7	68.9	99	<0.1	0.2	34.8
		4	97	<0.1	0.6	52.0	89	<0.1	0.4	89.0
		5	100	0.0	0.4	15.1	96	<0.1	0.3	39.8
	5	3	92	<0.1	0.8	79.0	100	0.0	0.2	7.7
		4	100	0.0	0.5	14.6	97	<0.1	0.2	21.4
		5	100	0.0	0.4	10.7	97	<0.1	0.2	27.4
	8	3	96	<0.1	0.8	62.2	100	0.0	0.1	15.7
		4	97	<0.1	0.7	45.4	94	<0.1	0.3	44.5
		5	100	0.0	0.4	14.9	97	<0.1	0.2	25.1
40	5	3	64	0.4	1.1	293.3	95	<0.1	0.2	63.6
		6	96	<0.1	0.4	40.3	89	<0.1	0.2	91.7
		9	97	<0.1	0.3	16.1	95	<0.1	0.2	64.7
	8	3	67	0.3	0.9	266.3	85	<0.1	0.2	119.2
		6	95	<0.1	0.4	59.1	88	<0.1	0.2	87.1
		9	100	0.0	0.2	6.6	97	<0.1	0.1	38.3
	12	3	55	0.4	1.9	341.4	74	<0.1	0.3	196.1
		6	98	<0.1	0.3	35.8	82	<0.1	0.2	134.6
		9	98	<0.1	0.2	9.3	95	<0.1	0.2	44.8
	16	3	53	0.3	0.9	385.5	68	0.1	0.2	227.8
		6	95	<0.1	0.4	66.4	72	<0.1	0.2	199.7
		9	100	0.0	0.2	8.9	89	<0.1	0.2	96.9
Sum/Avg.			4595	<0.1	0.5	41.9	4595	<0.1	0.2	36.4

Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M. Y., 2008. A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187 (3), 985–1032.

Azizoglu, M., Kirca, O., 1999. On the minimization of total weighted flow time with identical and uniform parallel machines. European Journal of Operational Research 113 (1), 91–100.

Azizoglu, M., Webster, S., 2003. Scheduling parallel machines to minimize weighted flowtime with family set-up times. International Journal of Production Research 41 (6), 1199–1215.

Bettayeb, B., Kacem, I., Adjallah, K. H., 2008. An improved branch-and-bound algorithm to minimize the weighted flowtime on identical parallel machines with family setup times. Journal of Systems Science and Systems Engineering 17 (4), 446–459.

Bruck, B. P., Iori, M., 2017. Non-elementary formulations for single vehicle routing problems with pickups and deliveries. Operations Research 65 (6), 1597–1614.

Bruno, J., Coffman, Jr., E. G., Sethi, R., 1974. Scheduling independent tasks to reduce mean finishing time. Communications of the ACM 17 (7), 382–387.

Chen, Z.-L., Powell, W. B., 1999. Solving parallel machine scheduling problems by column generation. INFORMS Journal on Computing 11 (1), 78–94.

Chen, Z.-L., Powell, W. B., 2003. Exact algorithms for scheduling multiple families of jobs on parallel machines. Naval Research Logistics (NRL) 50 (7), 823–840.

Côté, J.-F., Iori, M., 2018. The meet-in-the-middle principle for cutting and packing problems. INFORMS Journal on Computing 30 (4), 646–661.

Crauwels, H., Hariri, A., Potts, C., Van Wassenhove, L., 1998. Branch and bound algorithms for single-machine scheduling with batch set-up times to minimize total weighted completion time. Annals of Operations Research 83 (0), 59–76.

Crauwels, H., Potts, C., Van Wassenhove, L., 1997. Local search heuristics for single machine scheduling with batch set-up times to minimize total weighted completion time. Annals of Operations Research 70 (0), 261–279.

Delorme, M., Iori, M., Martello, S., 2016. Bin packing and cutting stock problems: Mathematical models and exact algorithms. European Journal of Operational Research 255 (1), 1–20.

Dunstall, S., Wirth, A., 2005a. A comparison of branch-and-bound algorithms for a family scheduling problem with identical parallel machines. European Journal of Operational Research 167 (2), 283–296.

Dunstall, S., Wirth, A., 2005b. Heuristic methods for the identical parallel machine flowtime problem with set-up times. Computers & Operations Research 32 (9), 2479–2491.

Dunstall, S., Wirth, A., Baker, K., 2000. Lower bounds and algorithms for flowtime minimization on a single machine with set-up times. Journal of Scheduling 3 (1), 51–69.

Elmaghraby, S. E., Park, S. H., 1974. Scheduling jobs on a number of identical machines. AIIE Transactions 6 (1), 1–13.

Feillet, D., 2010. A tutorial on column generation and branch-and-price for vehicle routing problems. 4OR 8 (4), 407–424.

Gavish, B., Graves, S., 1978. The traveling salesman problem and related problems. Operations Research Center, Massachusetts Institute of Technology Working Paper OR 078-78. URL `http://hdl.handle.net/1721.1/5363`

Ghosh, J. B., 1994. Batch scheduling to minimize total completion time. Operations Research Letters 16 (5), 271–275.

Graham, R., Lawler, E., Lenstra, J., Kan, A., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: P.L. Hammer, E. J., Korte, B. (Eds.), Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications. Vol. 5 of Annals of Discrete Mathematics. Elsevier, pp. 287–326.

Kim, D.-W., Kim, K.-H., Jang, W., Chen, F. F., 2002. Unrelated parallel machine scheduling with setup times using simulated annealing. Robotics and Computer-Integrated Manufacturing 18 (3), 223–231, 11th International Conference on Flexible Automation and Intelligent Manufacturing.

Kowalczyk, D., Leus, R., 2018. A branch-and-price algorithm for parallel machine scheduling using ZDDs and generic branching. INFORMS Journal on Computing, Forthcoming.

Kramer, A., Dell'Amico, M., Iori, M., 2018. Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines. CoRR abs/1808.10661, technical report available at: http://arxiv.org/abs/1808.10661.

Kramer, A., Subramanian, A., 2017. A unified heuristic and an annotated bibliography for a large class of earliness-tardiness scheduling problems. Journal of Scheduling, Forthcoming.

Liao, C.-J., Chao, C.-W., Chen, L.-C., 2012. An improved heuristic for parallel machine weighted flowtime scheduling with family set-up times. Computers & Mathematics with Applications 63 (1), 110–117.

Lübbecke, M. E., Desrosiers, J., 2005. Selected topics in column generation. Operations Research 53 (6), 1007–1023.

Macedo, R., Alves, C., de Carvalho, J. V., Clautiaux, F., Hanafi, S., 2011. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. European Journal of Operational Research 214 (3), 536–545.

Mason, A. J., Anderson, E. J., 1991. Minimizing flow time on a single machine with job classes and setup times. Naval Research Logistics (NRL) 38 (3), 333–350.

Monma, C. L., Potts, C. N., 1989. On the complexity of scheduling with batch setup times. Operations Research 37 (5), 798–804.

Mrad, M., Souayah, N., 2018. An arc-flow model for the makespan minimization problem on identical parallel machines. IEEE Access 6, 5300–5307.

Pessoa, A., Uchoa, E., de Aragão, M. P., Rodrigues, R., 2010. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. Mathematical Programming Computation 2 (3), 259–290.

Potts, C. N., Kovalyov, M. Y., 2000. Scheduling with batching: A review. European Journal of Operational Research 120 (2), 228–249.

Salazar-González, J.-J., Santos-Hernández, B., 2015. The split-demand one-commodity pickup-and-delivery travelling salesman problem. Transportation Research Part B: Methodological 75, 58–73.

Tseng, C.-T., Lee, C.-H., 2017. A new electromagnetism-like mechanism for identical parallel machine scheduling with family setup times. The International Journal of Advanced Manufacturing Technology 89 (5), 1865–1874.

Unlu, Y., Mason, S. J., 2010. Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. Computers & Industrial Engineering 58 (4), 785–800.

Valério de Carvalho, J., 1999. Exact solution of bin-packing problems using column generation and branch-and-bound. Annals of Operations Research 86 (0), 629–659.

van den Akker, J. M., Hoogeveen, J. A., van de Velde, S. L., 1999. Parallel machine scheduling by column generation. Operations Research 47 (6), 862–872.

Webster, S., 1997. The complexity of scheduling job families about a common due date. Operations Research Letters 20 (2), 65–74.

Webster, S., Azizoglu, M., 2001. Dynamic programming algorithms for scheduling parallel machines with family setup times. Computers & Operations Research 28 (2), 127–137.